Optimizing Sampling-based Entity Resolution over Streaming Documents

Christan Earl Grant[*] Daisy Zhe Wang[†]

Abstract

Increasingly, organizations have employed methods to understand unstructured text across the web. Entity resolution is used to identify mentions in large, streaming text corpora. Sampling-based entity resolution using Markov Chain Monte Carlo (MCMC) techniques guarantees convergence to a stationary distribution and can jump out of a local optimum. When performing entity resolution over streams of incoming data, the growing quantity of data amplifies two central issues. First, because the sampling process is random, many iterations are wasted attempting to resolve unambiguous entities. Second, the quadratic runtime for scoring entities becomes prohibitive for largest entities. Frequent streaming updates from the web exacerbate these difficulties. In this paper, we discuss the creation of a proposal optimizer, in the spirit of database optimizers. This optimizer observes the proposal updates to the entity resolution model then makes recommendations to improve the processing and storage of the model. We motivate the use of compression techniques to reduce the amount of processing when scoring MCMC updates proposal. We also discuss statistical early-stopping techniques for scoring entities. We describe our initial progress over a large entity resolution data set and how an optimizer can improve performance when processing entity resolution streams.

1 Introduction

Recently, an increasing number of organizations are tracking information across social media and the web. To this end, the National Institute of Standards hosted a three-year track to accelerate the extraction of information and construction of knowledge bases from streaming web resources [5]. This international contest highlighted the many difficulties of dealing with collecting unstructured data across the web. Across these efforts in this contest, we identify entity resolution as a major barrier to progress.

Entity resolution across text corpora is the task of identifying mentions within the documents that correspond to the same real-world entities. To construct knowledge bases or extract accurate information, entity resolution (ER) is a required step. This task is a notoriously computationally difficult problem. Using Markov Chain Monte Carlo (MCMC) techniques exchanges raw performance for a flexible representation and guaranteed convergence [7, 10, 13].

Processing streaming textual documents exacerbates two of the core difficulties of ER. The first difficulty is the computation of large entities, and the second is the excessive computation spent resolving unambiguous entities. Over time, the growing size of large entities makes keeping up with the incoming documents untenable. Optimization that touches these critical portions is wholly understudied. In this paper, we argue that compression and approximation techniques can efficiently decrease the runtime of traditional ER systems thus making them usable for streaming environment.

In sampling-based entity resolution, entities are represented as clusters of mentions. A proposal is made to move a random mention from a source entity to a random destination entity. The proposed state is scored and if it improves the global state, the new state is accepted. If the proposal does not improve the global state, the proposal may still be accepted with some small probability. This process is repeated until the state converges. Scoring the state of an entity cluster, through pairwise feature computation of the cluster mentions, is $O(n^2)$. For entity clusters larger than 1000 mentions, calculating the score for each proposal can become prohibitively expensive.

Wick et al. present an entity resolution technique that uses a tree structure to organize related entities to reduce the amount of work performed in each step [13]. During each proposal, this approach avoids the pairwise comparison by restricting model calculation to the top nodes of the hierarchy. This approach can avoid massive amounts of computation by performing organizing

[*]Datascience Research Lab, Computer & Information Science & Engineering Department, University of Florida, Gainesville, Florida; cgrant@cise.ufl.edu

[†]Datascience Research Lab, Computer & Information Science & Engineering Department, University of Florida, Gainesville, Florida; daisyw@cise.ufl.edu

the known sets of mentions. This discriminative tree structure is a type of *compression*.

Singh et al. present a method of efficiently sampling factors to reduce the amount of work performed when computing features [12]. They observe that many factors are redundant and do not need to be computed when calculating the feature score. They use statistical techniques to estimate the computed feature scores with a user-specified confidence. This approach can be categorized as *early stopping* for feature computation.

There is no one size fits all sampling algorithm [9]; each of these methods, compression and early stopping, has drawbacks. Compression may slow down insertion speed and requires extra book keeping to keep to organize the data structure. Early stopping is not always precise and adding extra conditionals in the metropolis hastings loop structure slows computation. Applying each technique at appropriate times can remove pain points and accelerate the entity resolution process.

In this paper, we discuss our initial work towards the design of an optimizer that modifies the sampling-based collective entity resolution process to improve sampling performance. Static parameters for evaluating entity resolution rarely hold for the lifetime of streaming processing task. The optimizer, in the spirit of the *eddy* database query optimizer [1], dynamically examines the current state of each proposal and suggests methods for evaluating proposals and structuring entities. We train a classifier to decide when the sampling process should use early stopping. Additionally, we use training data to decide when is the best time for a particular entity to be compressed. This is done with negligible book keeping. We make the following contributions:

- We identify several techniques to speed up sampling past a natural baseline.
- We create rules and techniques for an optimizer to choose parameters and methods at run time.
- We empirically evaluate these methods over a large data set.

We recognize that optimizers can also apply to many different long running machine learning pipeline. Figure 1 depicts that the optimizer supervises the machine learning model. The optimizer determines the methods of processing the streaming updates of the model. As future work, we plan to create a full optimizer to study performance improvements on long running machines learning tasks.

The outline of the paper is as follows. In Section 2, we give a introduction to factor graph models and entity resolution. In Section 3, we further discuss the statistics that an optimizer for entity resolution can use. In

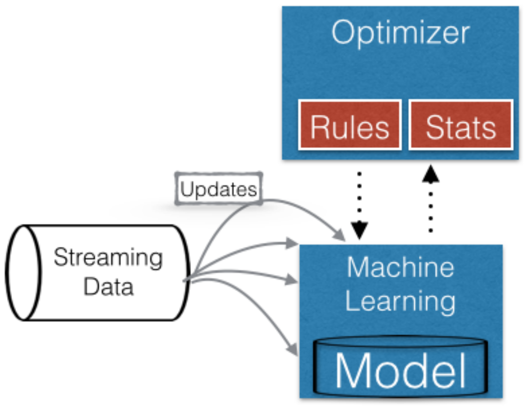

Figure 1: The high-level interaction of the optimizer. As streaming data updates pass to the machine learning model, the optimizer recommends the best algorithms to update the model. Entity resolution is an example of a model that needs to be frequently updated with new data.

Sections 4 and 5, we discuss the implementation of the optimizer. Finally, in Section 6, we examine the benefits by testing early stopping and compression over a synthetic and a popular real world entity resolution data set.

2 Background

Factor graphs are a pairwise formalism for expressing arbitrarily complex relationships between random variables [6]. A factor graph $\mathcal{F} = \langle \mathbf{x}, \psi \rangle$, contains a set of random variables $\mathbf{x} = \{x_i\}_1^n$ and factors $\boldsymbol{\psi} = \{\psi_i\}_1^m$. Random variables are connected to each other through factors. Factors are a mapping between one or more variables and a real-valued score.

The probability of a setting ω among the set of all possible settings Ω occurring in a factor graph is given by a probability measure:

$$\pi(\omega) = \frac{1}{Z} \sum_{x \in \omega} \prod_{i=1}^{m} \psi_i(x^i), \quad Z = \sum_{\omega \in \Omega} \sum_{x \in \omega} \prod_{i=1}^{m} \psi_i(x^i)$$

where x^i is the set of random variables that neighbor the factor $\psi_i(\cdot)$ and Z is the normalizing constant.

Exact inference over complex factors graphs is computationally expensive because it involves computing the normalizing constant. Therefore, it is popular for researchers to use Markov Chain Monte Carlo (MCMC) approximation techniques to estimate the probability of settings. In particular, for large and dense factor graphs MCMC Metropolis Hastings (MH) has been shown to be a scalable technique for inference calculation [10].

Cross-Document entity resolution, resolving entities

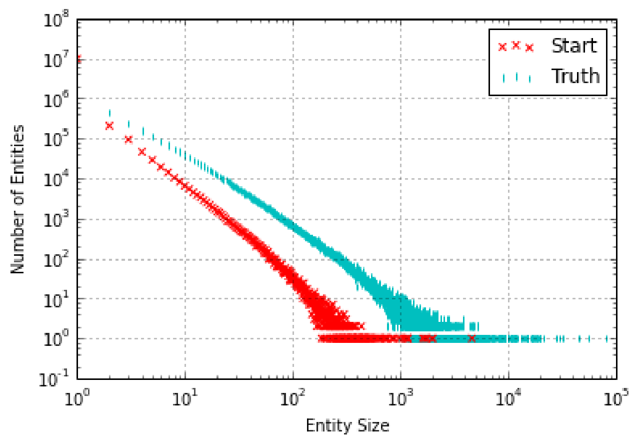

Figure 2: A distribution of entity sizes from the wiki links corpus [11] with an initial start and the truth.

across document borders, is usually several orders of magnitude smaller when compared to within document entity resolution. In large text corpora, the size of entities follows the power law [11]. For example, Figure 2 is a generated data set containing 40 million mentions and 3 million entities over 11 million web pages. As documents and mentions are incrementally streamed through, the scale problem becomes a critical issue.

The mentions on disk can be represented as a large array of identifiers. Entities are a collection of mentions and can be represented as such. In the worst case there is an equal number of entities and mentions. This means each mention is its own individual entity. In the other extreme, all the mentions may be a part of the same entity. For streaming entity resolution, mentions within documents must be matched to the existing set of entities [8]. In this paper, we assume the entity set is initialized by grouping the most similar mentions; new mentions are assigned to the closed match.

To compute the score at each step, the number of comparisons is proportional to the number of pairwise factors between mentions. The pairwise factors are weighted functions such as approximate string matches, token overlap, n-gram matches. There are additional cluster-wide features calculated at each step. Such features include functions to check whether all mentions in a cluster share the same token. For clusters larger than 1000 mentions, calculating scores of the model becomes extremely expensive. Performing sophisticated techniques over smaller clusters also adds extra overhead. In this paper, we examine the trade-off of selecting methods to accelerate the feature computation process.

3 Accelerating Entity Resolution

In this section, we discuss the acceleration in MCMC-MH sampling for entity resolution. We then motivate how we believe gains can be achieved given using compression, sampling acceleration methods and optimizers. We use a large real-world corpus for a motivating example.

The two issues we are investigating are as follows: First, given a source entity, destination entity and the mention (e_s, e_d, m), which method can score the proposal in the *least amount of time*? Secondly, after the proposal is calculated, should we compress the entity structure? The optimizer will *decide* when to use each technique.

The total size of all entities in the traditional representation is:

$$\text{sizeof}(\mathcal{E}) = \sum_i c + (\text{sizeof}(\text{int}) * |e_i|), \tag{3.1}$$

where *sizeof* is an abstract function to compute the size of the containing object, c is a class constant and $|e_i|$ is number of mentions in the entity.

There are many compression techniques, one being to only keep mentions that have a unique representation inside entities. That is, if any mention token is a duplicate, we remove it. This compressed total entity size is:

$$\text{sizeof}(\mathcal{E}_{\text{compressed}}) = \sum_i c + (\text{sizeof}(int) * \#e_i), \tag{3.2}$$

where $\#e_i$ is the cardinality of the mention tokens in entity e_i. We note that when the $\#e_i \ll |e_i|$, it may be worth compressing the entity e_i.

In Figure 2, 45% percent of entities are smaller that 100 mentions in size. Additionally, 82% percent of entities contain less than 1000 mentions. These numbers suggest that at times we we can take advantage of the redundancy within large entities by compressing them. We investigate the wiki links corpus further in Section 6.1.

In addition, Figure 2 shows that there is an order of magnitude difference between the sizes of initial entities and the true entity sizes. The entities were initialized by exact string match, a common initialization scheme. This difference gives us some intuition of the trends of the entity resolution process. Additionally, this suggest that there are several distinct representations of entities During entity resolution the sizes of entities can expect to grow by an order of magnitude in size while the total number of smaller entities will decrease. We can use this property to track the growth and change of entity sizes over time to understand how to process a particular grouping of entities.

4 Algorithms

In this section, we will describe simple algorithms for entity sampling and entity simple compression. After introducing the compression and approximation techniques we discuss how an optimizer can be designed to improve the overall sampling time.

The baseline method performs pairwise comparisons by iterating over the mentions using the order on disk. The mentions ids are used to extract the contextual information of each mention from a database. This is the traditional method of computing the pairwise similarity of two clusters. This method results in simple code so modern compilers are able to perform extreme optimizations such as loop unrolling.

Confidence-based scoring method performs uniform samples of the mentions from the source and destination entities clusters during scoring. This method measures the confidence of the calculated pairwise samples and stops when the confidence of a score exceeds a threshold of 0.95. This is a simplified version of the sampling uniform sampling method described by Singh et al. [12].

The code to collect statistics is shown in Algorithm 3. The `add` function shows how and what statistics are recorded when each new mention is added. Notice `themax` and `themin` are variables in the Stats class that store the current maximum and minimum. The current sum, running mean are also updated with each new value added. The current implementation assumes the values from the pairwise factors follow a Gaussian distribution; the model in Singh et al. make the same assumption [12].

As entity sizes grow, we can expect to see many repeats of the same or very similar mentions. Reducing the entity size will shrink the effective memory footprint of entities. This is important for long running collection of entities. Run-length encoding is the simplest method for compressing entities. This method compresses the near duplicate mentions. A canonical mention is chosen along each exact duplicate and a counter map records the number of duplicates that are represented. The compression rates become large for mention clusters with many duplicate.

5 Optimizer

When before calculating the MCMC-MH proposal there are several decision we can make that will affect the runtime and accuracy of the algorithm. At each step we may: (1) approximate the calculation of the entity states; (2) update an entity structure to a compressed format; (3) skip the calculation of the proposal and directly accept or reject. These decisions can be made by observing several features of a source entity, destination entity and a source mention. We enumerate a small set

```
void Stats::add(long double x) {
  themax = MAX(themax,x);
  themin = MIN(themin,x);
  _sum += x;
  ++n;
  auto delta = x - mean;
  mean += (delta / n);
  M2 = M2 + delta * (x - mean);
}

double Stats::variance (void) const {
  if (n > 2)
  return M2 / (n-1);
  else
  return 0.0;
}
```

Figure 3: Sample code from the stats showing how running statistics are recorded and how the variance can be computed.

of features that can yield information to help us decide how the entity structure should be changed.

The decision to compress an entity takes four main points into consideration. First, the time it takes to compress the entity (C_{time}). For example, if the time it takes to compress an entity is the same as the time it takes to reach an answer in the uncompressed format, then compression is superfluous. Second, it is important to consider the spaced saved in memory and the amount of additional entities that do not have to be fetched from disk and can now fit in memory (C_{space}). Third, we need to know how active an entity has been (C_{activity}). That is, how many additions or subtractions this entity has seen over a long period of time. This information is helpful in understanding the likelihood this entity will be requested for another addition or subtraction. (Modifying entities clusters causes them to block.) Last, we retain the activity of an entity over a recent, short period of time (C_{velocity}). This information lets us know whether it is smart for this entity to take the time out to for compression while other mentions may be attempting an insertion or removal.

At each proposal step the decision made should maximize the *utility*. Utility of the decision is a numeric score to represent the gain performing the proposal calculation. The utility value is a real number ranged from $(-\infty, \infty)$. A formal model for utility is as follows:

$$U = C_{\text{time}} + C_{\text{space}} + C_{\text{activity}} + C_{\text{velocity}}$$

Collecting statistics to measure utility is can incur a significant overhead. Not every decision in the optimizer

Technique	Compression	Early Stopping	Overhead
Baseline	No	No	None
Confidence-based [12]	No	Yes	Medium
Discriminative Tree [13]	Yes	No	Large
Run-Length Encoding	Yes	No	Small

Table 1: A table of the techniques to improve the sampling process and each is classified by how they affect sampling.

needs to be decided automatically. We can use some simple principles to estimate the utility at each point. In the next section we, examine an entity resolution data set and get some intuition for the development of the optimizer.

6 Implementation

In this section, we first describe the wiki link data set we use for experiments. Following, we present a micro benchmark to validate our investigation of entity approximation and compression. We then discuss the implementation of the compression and approximation techniques over a large real-world cross-document entity resolution corpus.

6.1 Wiki Link Corpus The wiki link corpus is the largest fully labeled cross-document entity resolution data set to date [11]. When downloaded, the data set contains 40 million mentions and almost three million entities — it is a compressed 180 GBs of data. The wiki link corpus was created by crawling pages across the web and extracting anchor tags that referenced Wikipedia articles. Each page contains multiple multiple mentions of different types. The Wikipedia articles act as the truth for each mention. Although manually constructed and not without its biases, this is the largest, fully-labeled entity resolution data set over web data that we could find (at the time of preparation).

6.2 Micro benchmark To increase our intuition of early stopping techniques we simulated the MCMC proposal processes. We hypothesise that a range of values exist, where performing the baseline cluster sampling would be faster than early stopping methods. We arrange entity clusters of increasing size and we compute the time (in clock ticks) each proposal takes to compute the arrangement of the clusters. The data in the clusters are distributed uniformly for this experiment and each cluster point was 5 dimensional. For the baseline cluster score computation we used a pairwise calculated of the average cosine distance with and without the mention. To compute early stopping we set a confidence threshold to 0.8 and the early stopping

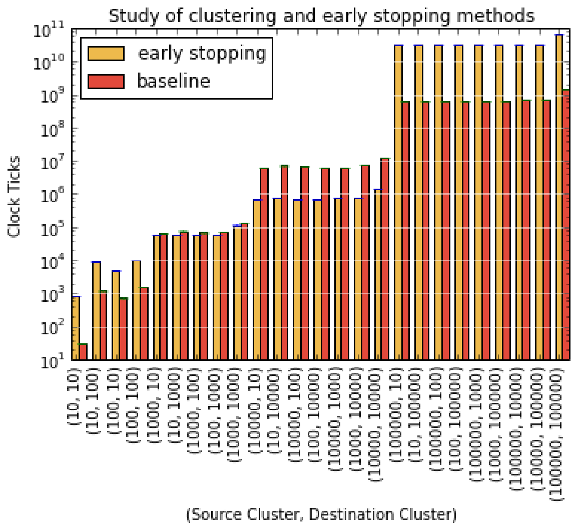

Figure 4: Comparison of baseline verses early stopping methods.

code stopped computation when the predicted error was under 20%. There was no difference in the proposal choices of the baseline method or the early sorting method.

The simulations were developed in `GNU C++11` and compiled with `g++ -O3`. The CPU was an 8 core Intel i7 with 3.2 GHz and 12 GBs of Memory. Each arrangement was run 5 times and results averages.

Early stopping or baseline. We first determine when early stopping approaches from proposal scoring is beneficial. For this result we compare the base like proposal evaluator with a confidence-based scorer for varying entity sizes. The result of this experiment is summarized in Figure 4. The x-axis is the number of mentions in the source and destination cluster for each proposal. The y-axsis is the number of clock ticks on a log-scale.

We observe that for proposals with less than 100 and 1000 source and destination mentions, the performance of the baseline proposer is better than or almost equal to that of the more sorted early stopping method. For proposals that contain an entity cluster with 10000

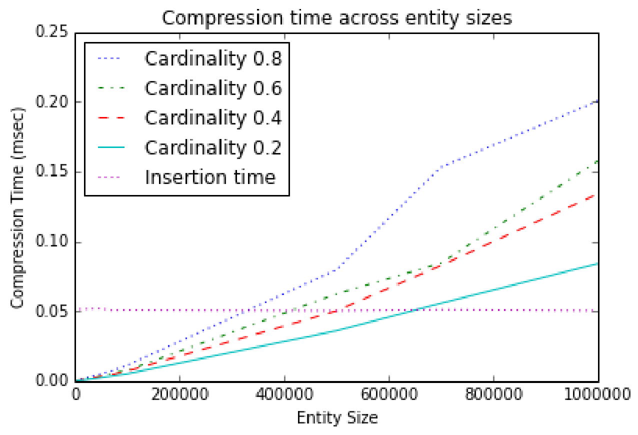

Figure 5: The time for compression for varying entity sizes and cardinalities.This is compared with line representing the time it take to make 100K insertions.

mentions the early stopping method performs significantly better than the baseline method.

Surprisingly, the baseline proposals for for entities clusters containing $100K$ mentions performed over an order of magnitude better than the early stopping method.

The optimization found in predictable code paths make simple implementations like the baseline method attractive for small cluster sizes and very large clusters sizes. In addition, 82% of the entities in the truthed wiki links data sets are less that 1000 mentions in size and 45% of the entities contain less than 100 mentions.

The results of the micro benchmark suggests that different proposal estimation techniques are useful at different times. Note that for these techniques a small constant amount of book keeping space is required to perform early stopping.

Insertion vs Compressions Time. Compressing an entity is an expensive operation. When compression and entity, it must be locked to prevent any concurrent access. In order to choose the best times to compress an entity cluster in this micro benchmark we look at the time to compression entity of different cardinalities and compare them to the time it takes to insert entities. Using a synthetic data set we generated entities of varying sizes and cardinality. This experiment is shown in Figure 5.

Cardinality number is a ratio of duplicates in the data set. For example, Cardinality 0.8 means 8 of 10 items in the data set are duplicates. The graph shows that in the time it take to compress entities of about 300K the sampler could make 100K samples. We can conclude from these result that compressing large entities is expensive should only be done if the cluster is prohibitively large and not popular.

Cardinality estimation for millions of entities is a significant overhead. Tracking cardinalities simultaneously for each entity, even using small probabilistic sketches such as Hyperloglog [4] become prohibitive for large amounts of entities. By the time the cardinality of an entity needs to be monitored for possible compression, that entity might as well be compressed. We are continuing to look for lighter weight cardinality estimators for millions of mentions so decisions can quickly be made.

7 Summary

In this paper, we describe an initial approach for optimizing sampling for the entity resolution process. We begin to develop an optimizer that attacks two major limitations, the size of the entities and the redundant computation. This paper motivated the need for the optimizer and examined the feasibility of its treation. We plan to implement the full optimizer over a large, streaming corpus, with resolved entities. We hope to soon have a fully resolved TREC streamcorpus[1] and examine the performance of the optimizer of that large data set. Additionally, we hope to compare results with enterprise ER systems such as WOO [2].

8 Acknowledgements

This work was partially supported by DARPA under FA8750-12-2-0348-2 (DEFT/CUBISM) and an NSF Graduate Research Fellowship, Grant DGE-0802270.

References

[1] R. Avnur and J. M. Hellerstein. Eddies: Continuously adaptive query processing. In *ACM SIGMOD Record*, volume 29, pages 261–272. ACM, 2000.

[2] K. Bellare, C. Curino, A. Machanavajihala, P. Mika, M. Rahurkar, and A. Sane. Woo: A scalable and multi-tenant platform for continuous knowledge base synthesis. *Proc. VLDB Endow.*, 6(11):1114–1125, Aug. 2013.

[3] J. Dalton, J. R. Frank, E. Gabrilovich, M. Ringgaard, and A. Subramanya. Fakba1: Freebase annotation of trec kba stream corpus, version 1 (release date 2015-01-26, format version 1, correction level 0), January 2015.

[4] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. *DMTCS Proceedings*, 0(1), 2008.

[1]After acceptance the `http://trec-kba.org/kba-stream-corpus-2014.shtml` was linked to Freebase and is now available for researchers [3].

[5] J. R. Frank, S. J. Bauer, M. Kleiman-Weiner, D. A. Roberts, N. Tripuraneni, C. Zhang, C. Re, E. Voorhees, and I. Soboroff. Evaluating stream filtering for entity profile updates for trec 2013 (kba track overview). Technical report, DTIC Document, 2013.

[6] D. Koller and N. Friedman. *Probabilistic graphical models: principlese and techniques*. MIT press, 2009.

[7] A. Mccallum and B. Wellner. Toward conditional models of identity uncertainty with application to proper noun coreference. In *In NIPS*, pages 905–912. MIT Press, 2003.

[8] D. Rao, P. McNamee, and M. Dredze. Streaming cross document entity coreference resolution. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, COLING '10, pages 1050–1058, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[9] D. Sculley and C. E. Brodley. Compression and machine learning: A new perspective on feature space vectors. In *Data Compression Conference, 2006. DCC 2006. Proceedings*, pages 332–341. IEEE, 2006.

[10] S. Singh, A. Subramanya, F. Pereira, and A. McCallum. Large-scale cross-document coreference using distributed inference and hierarchical models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 793–803. Association for Computational Linguistics, 2011.

[11] S. Singh, A. Subramanya, F. Pereira, and A. McCallum. Wikilinks: A large-scale cross-document coreference corpus labeled via links to Wikipedia. Technical Report UM-CS-2012-015, University of Massachusetts, Amherst, 2012.

[12] S. Singh, M. Wick, and A. McCallum. Monte carlo mcmc: efficient inference by approximate sampling. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1104–1113. Association for Computational Linguistics, 2012.

[13] M. Wick, S. Singh, and A. McCallum. A discriminative hierarchical model for fast coreference at large scale. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers - Volume 1*, ACL '12, pages 379–388, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.